

`rstudio::conf(2020L)`

# List-columns in `data.table`



Tyson S. Barrett

Absolutely Amazing Artwork by Allison Hurst

# Why list-columns?

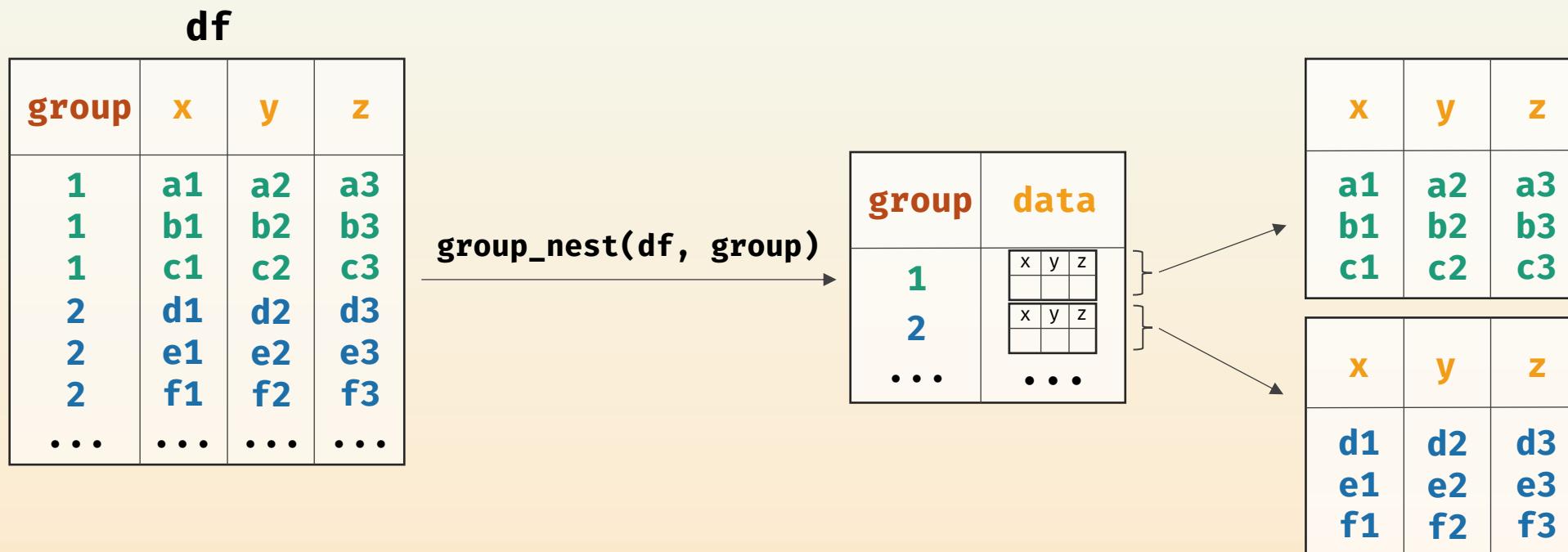
Can make data manipulations  
safer

Cognitively easier to  
understand complex  
data

Memory efficient  
(fewer redundancies)

Has several functions in  
the **#tidyverse**

# What are list-columns?



# Why `data.table` ?

Concise syntax

Fast speed

Memory efficient

Careful API lifecycle management

Community

Feature rich

# Why `data.table` ?

Concise syntax

Fast speed

Memory efficient

Careful API lifecycle management

Extensible

Feature rich

# Why `data.table` ?

Concise syntax

Fast speed

Memory efficient

Careful API lifecycle management

Feature rich

`dt[i, j, by]`

The diagram shows the `dt[i, j, by]` code with three arrows pointing to specific words:

- A blue arrow points to the word `i`, which is associated with the word "filter".
- A green arrow points to the word `j`, which is associated with the word "mutate".
- An orange arrow points to the word `by`, which is associated with the word "group\_by".

# Why `data.table` ?

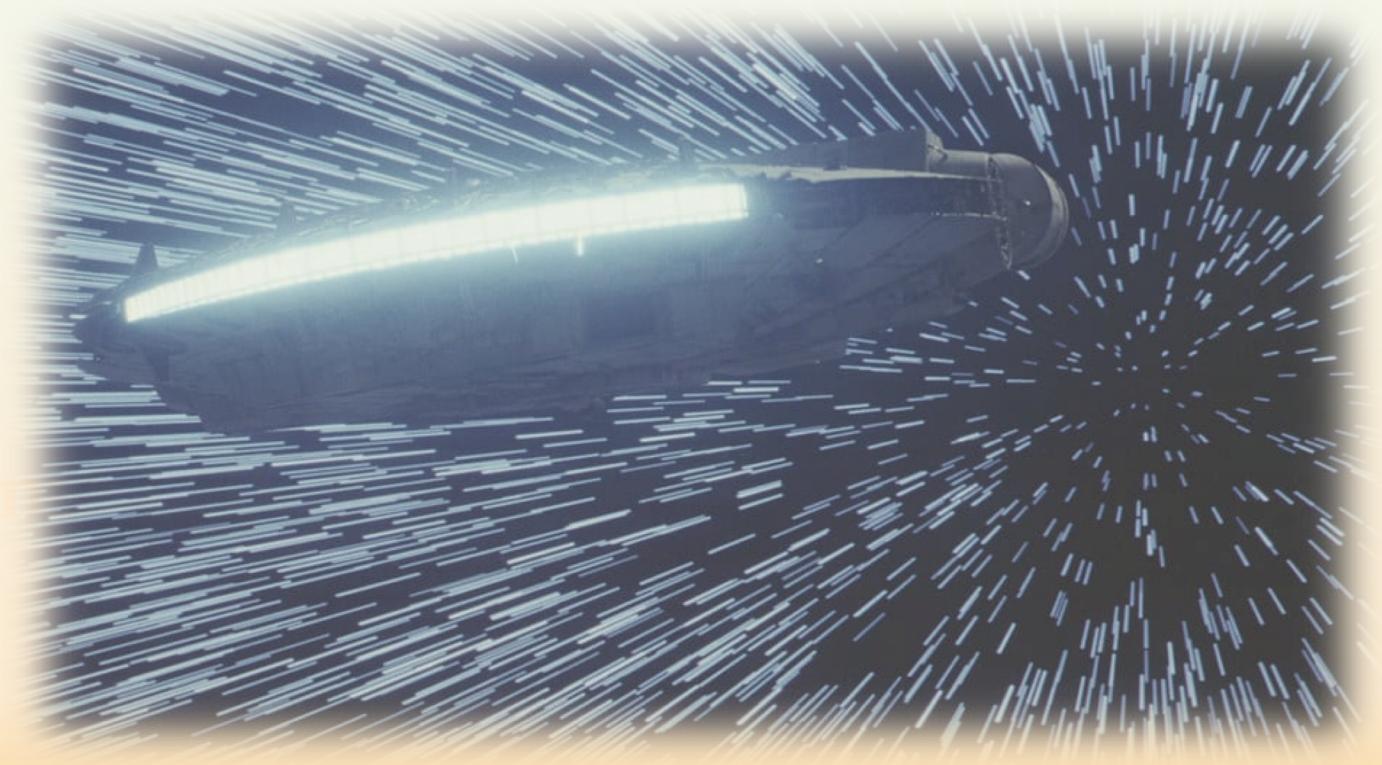
Concise syntax

Fast speed

Memory efficient

General API interface

Feature rich



# Why `data.table` ?

Concise syntax

Fast speed

Memory efficient

Careful API design

Feature rich

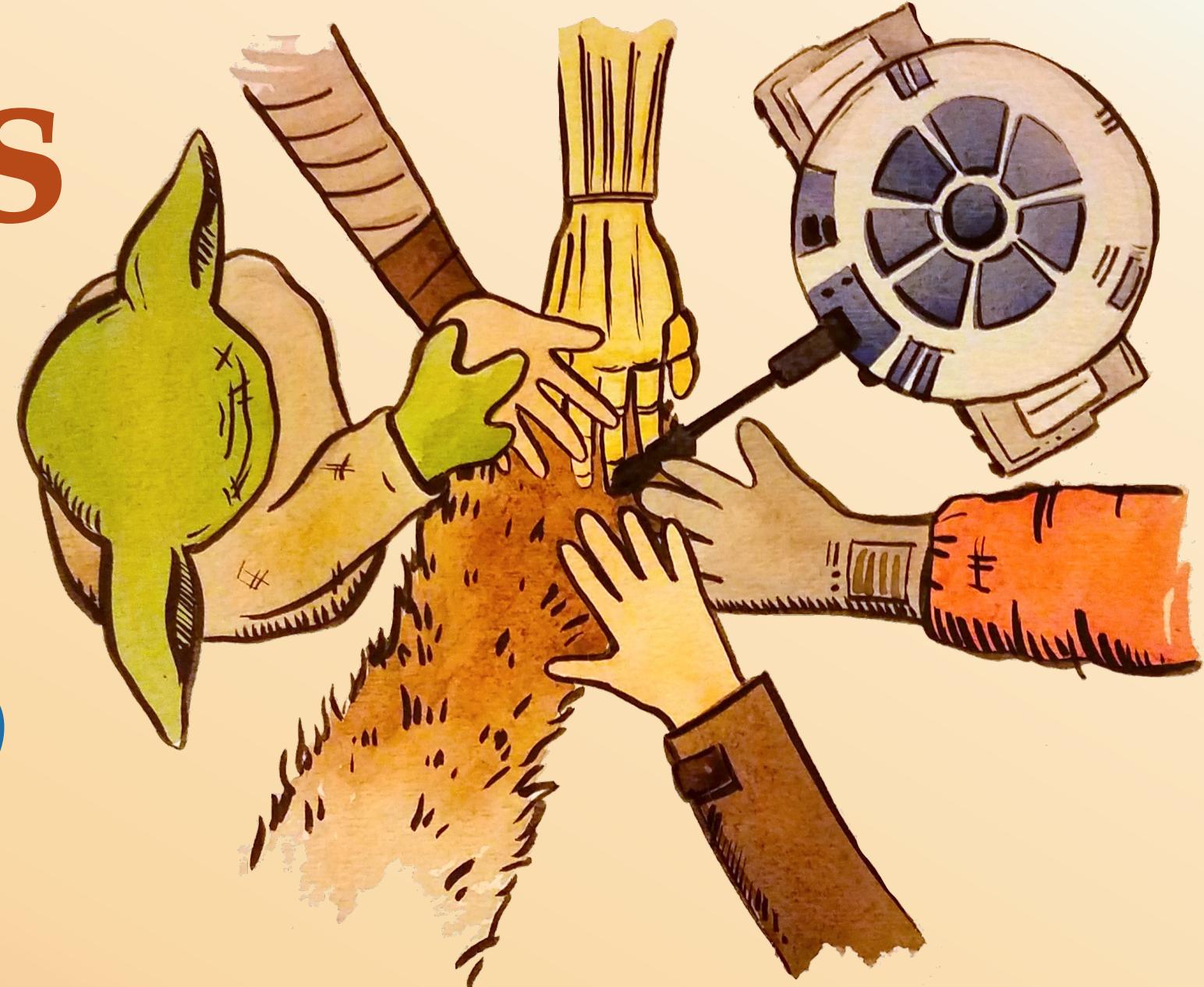
<https://h2oai.github.io/db-benchmark/>

## Summing by group

Input table: 1,000,000,000 rows x 9 columns ( 50 GB )

■	data.table	1.12.9	2019-12-12	1339s
■	(py)datatable	0.10.0a0	2019-11-24	2682s
■	dplyr	0.8.3	2019-12-05	timeout
■	pandas	0.25.3	2019-12-07	out of memory
■	spark	2.4.4	2019-12-05	not yet implemented
■	dask	2.9.0	2019-12-07	timeout
■	DataFrames.jl	0.20.0	2019-12-09	out of memory
■	ClickHouse	19.16.2.2	2019-12-09	CH server crash
■	cuDF	0.10.0	2019-12-05	out of memory
■	Modin		see README	pending

Let's  
Team  
Up



# The Grammar

**Nest** making a list-column of data tables or tibbles

**Hoist** unnesting list-columns that have vectors in them

**Unnest** unnesting list-columns that have data tables or tibbles in them

# Typing R Code

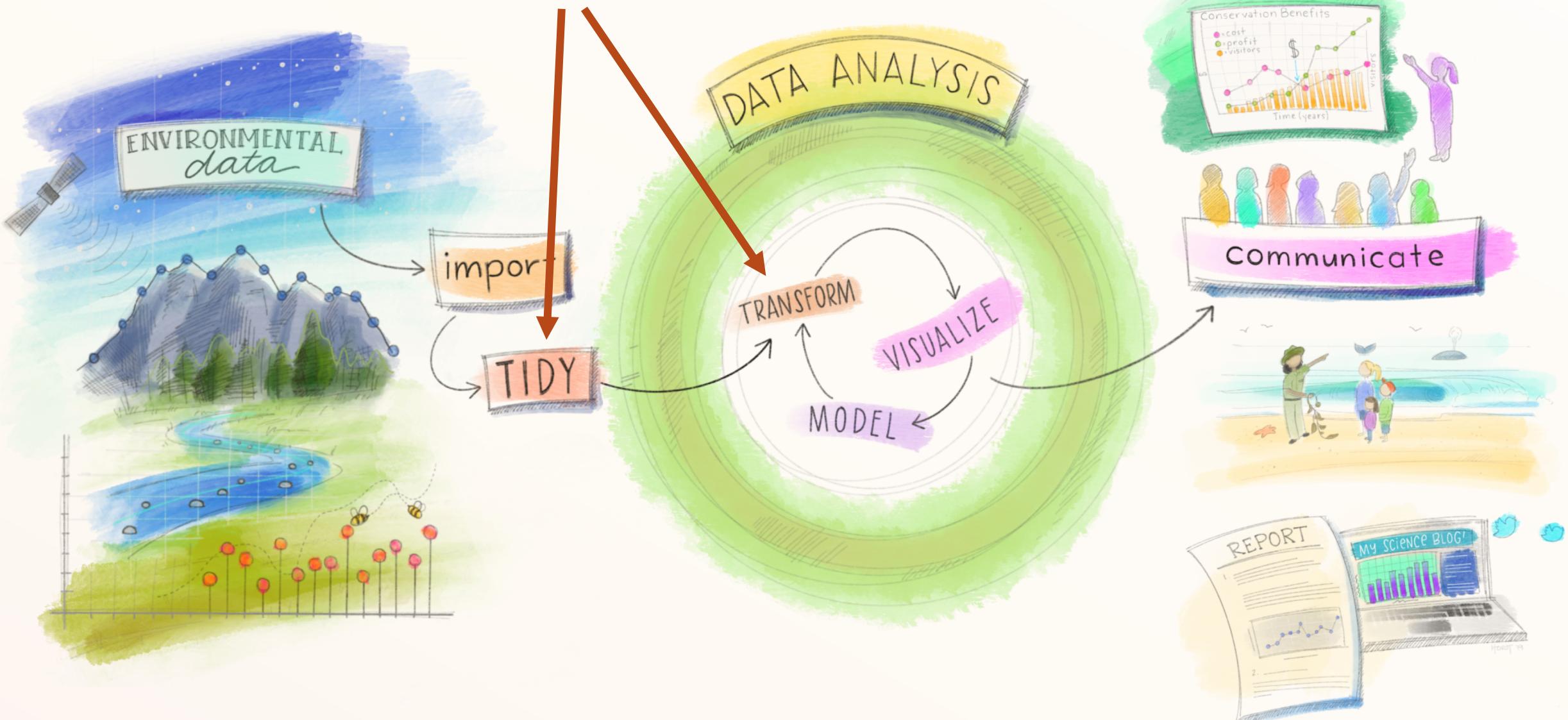
```
library(dplyr)
library(data.table)

dt_starwars <-
  as.data.table(dplyr::starwars)
```

# Typing R Code

```
##   name                      species homeworld films
##   <chr>                     <chr>   <chr>    <list>
## 1 Luke Skywalker            Human   Tatooine  <chr [5]>
## 2 C-3PO                      Droid   Tatooine  <chr [6]>
## 3 R2-D2                      Droid   Naboo    <chr [7]>
## 4 Darth Vader                Human   Tatooine  <chr [4]>
## 5 Leia Organa                Human   Alderaan  <chr [5]>
## 6 Owen Lars                  Human   Tatooine  <chr [3]>
## 7 Beru Whitesun lars        Human   Tatooine  <chr [3]>
## 8 R5-D4                      Droid   Tatooine  <chr [1]>
## 9 Biggs Darklighter          Human   Tatooine  <chr [1]>
## 10 Obi-Wan Kenobi             Human   Stewjon  <chr [6]>
## # ... with 77 more rows
```

# You are here



# Typing R Code

`tidyfast` packages up the functions that use `data.table`

The variable `films` is currently a list-column of type: `<char>`  
We can hoist that column so we can use each film

```
library(tidyfast)
```

```
films <- dt_hoist(dt_starwars,  
                   films)
```

The data table

The list-column

# dt\_hoist()

The bulk of the work  
is just two lines  
*(everything else is mostly checks  
and formatting)*

```
dt_hoist.default <- function(dt_, ...){  
  if (isFALSE(is.data.table(dt_)))  
    dt_ <- as.data.table(dt_)  
  
  pasted_dots <- paste(substitute(list(...)))[-1L]  
  classes <- sapply(dt_, class)  
  typeofs <- sapply(dt_, typeof)  
  v.names <- names(classes)  
  keep <- v.names[classes != "list" & typeofs != "list"]  
  drop <- v.names[classes == "list" | typeofs == "list"]  
  drop <- drop[!drop %in% pasted_dots]  
  keep <- keep[!keep %in% pasted_dots]  
  keep <- paste(keep, collapse = ",")  
  cols <- substitute(unlist(list(...)), recursive = FALSE)  
  
  if (length(drop) > 1)  
    message("The following columns were dropped because ",  
           "they are list-columns (but not being hoisted): ",  
           paste(drop, collapse = ", "))  
  
  dt_ <- dt_[, eval(cols), by = keep]  
  dt_ <- .naming(dt_, substitute(list(...)))  
  dt_  
}
```

# Typing R Code

Create a variable called `data` with all the data associated with each film

```
nested <- dt_nest(films, films)
nested[]
```

The data table  
The list-column

# Typing R Code

```
##     films          data
##     <char>         <list>
## 1: A New Hope    <data.table>
## 2: Attack of the Clones <data.table>
## 3: Return of the Jedi   <data.table>
## 4: Revenge of the Sith   <data.table>
## 5: The Empire Strikes Back <data.table>
## 6: The Force Awakens    <data.table>
## 7: The Phantom Menace    <data.table>
```

# dt\_nest()

Again, the bulk of the work is just two lines

.SD is a placeholder for the whole data table, excluding the keyby variable(s)

keyby sorts the data using data.table's super fast sorting

```
dt_nest.default <- function(dt_, ..., .key = "data"){

  # change to data.table
  if (isFALSE(is.data.table(dt_)))
    dt_ <- as.data.table(dt_)

  # groups
  by <- substitute(list(...))

  # call to data.table
  dt_ <- dt_[, list(list(.SD)), keyby = eval(by)]
  setnames(dt_, old = "V1", new = .key)
  dt_

}
```

# Typing R Code

```
revenue <- tibble::tribble(  
  ~films, ~revenue,  
  "A New Hope", 775.4,  
  "The Empire Strikes Back", 538.4,  
  "Return of the Jedi", 475.1,  
  "The Phantom Menace", 1027.0,  
  "Attack of the Clones", 649.4,  
  "Revenge of the Sith", 848.8,  
  "The Force Awakens", 2068.2  
) %>%  
as.data.table()
```

```
nested <- nested[revenue, on = "films"] # join on films  
nested[]
```

# Typing R Code

```
##   films          data      revenue
##   <char>        <list>    <num>
## 1: A New Hope  <data.table> 775.4
## 2: The Empire ... <data.table> 538.4
## 3: Return of ... <data.table> 475.1
## 4: The Phantom ... <data.table> 1027.0
## 5: Attack of ... <data.table> 649.4
## 6: Revenge of ... <data.table> 848.8
## 7: The Force ... <data.table> 2068.2
```

# Typing R Code

Get some information from the data column (the list-column)  
*(Could use dplyr here as well!)*

```
prop_female <-  
  films_unnest[, prop_genders :=  
    purrr::map(data, ~dt_count(.x, gender))]  
  
films_unnest <- dt_unnest(prop_female, prop_genders)  
films_unnest[]
```

# Typing R Code

```
##   films          revenue gender      N
##   <char>        <num>  <char>  <int>
## 1: A New Hope    775.4  <NA>     3
## 2: A New Hope    775.4  female    2
## 3: A New Hope    775.4  hermaphrodite 1
## 4: A New Hope    775.4  male     12
## 5: The Empire ... 538.4  <NA>     2
## 6: The Empire ... 538.4  female    1
## 7: The Empire ... 538.4  male     12
---
```

From here, we can get the proportion of females to all characters and check the relationship between proportion and revenue

# dt\_unnest()

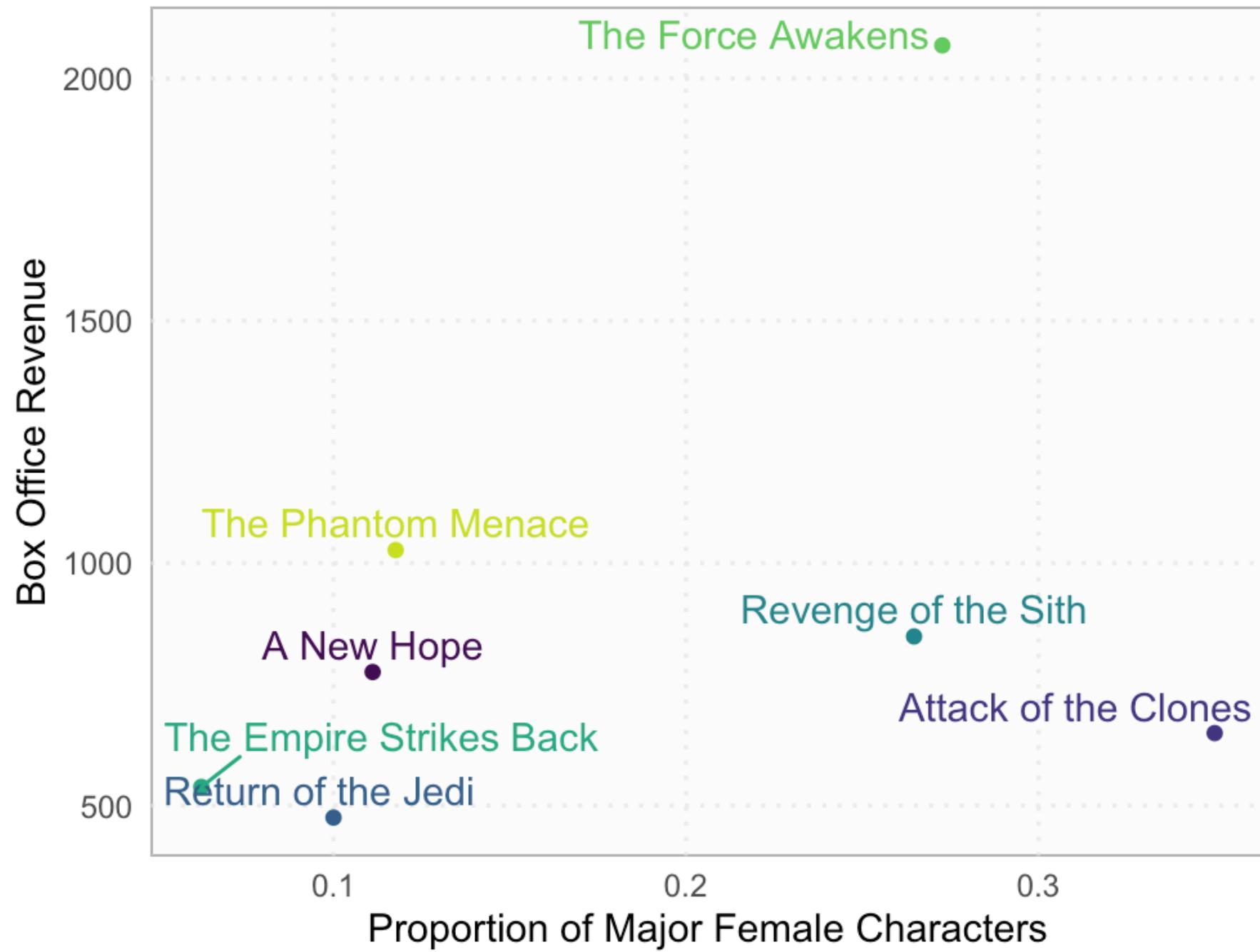
This one needed a few more lines of code but still only a few

The `..others` helps `data.table` to know to look outside of the table for the information (working with NSE)

`rbindlist` is `data.table`'s super fast row binder

```
dt_unnest.default <- function(dt_, col, ...){  
  if (isFALSE(is.data.table(dt_)))  
    dt_ <- as.data.table(dt_)  
  
  col     <- substitute(col)  
  keep    <- substitute(alist(...))  
  names   <- colnames(dt_)  
  others  <- names[-match(paste(col), names)]  
  rows    <- sapply(dt_[[paste(col)]], nrow)  
  
  if (length(keep) > 1)  
    others <- others[others %in% paste(keep)[-1]]  
  
  others_dt <- dt_[, ..others]  
  classes   <- sapply(others_dt, typeof)  
  keep       <- names(classes)[classes != "list"]  
  others_dt <- others_dt[, ..keep]  
  others_dt <- lapply(others_dt, rep, times = rows)  
  
  dt_[, list(as.data.table(others_dt), rbindlist(eval(col)))]  
}
```

# Results



Data from Statista and SWAPI.

# Cool, cool...

# But what about performance?!

```
#> # A tibble: 2 × 3  
#>   expression    median mem_alloc  
#>   <chr>        <bch:tm> <bch:byt>  
#> 1 dt_nest      3.16ms   2.88MB  
#> 2 group_nest   4.79ms   2.54MB
```

## Nesting

## Unnesting

```
#> # A tibble: 2 × 3  
#>   expression    median mem_alloc  
#>   <chr>        <bch:tm> <bch:byt>  
#> 1 dt_unnest    4.7ms    5.49MB  
#> 2 unnest       12.6ms   8.47MB
```

# Takeaways

**data.table** syntax can work with **tidyverse** grammar

**data.table** can do things quickly and efficiently

The **grammar** is important to communicate what is happening to the data

**data.table** and the **tidyverse** are not rivals, but rather are **complimentary**

# Links and Such

Tyson S. Barrett



[tyson.barrett@usu.edu](mailto:tyson.barrett@usu.edu)



[tysonbarrett.com](http://tysonbarrett.com)



[@healthandstats](https://twitter.com/healthandstats)



[github.com/tysonstanley](https://github.com/tysonstanley)



Slides at [tysonbarrett.com/teaching](http://tysonbarrett.com/teaching)